※ 코드의 양을 줄이기 위해 대부분의 예제 프로그램에서 에러 처리가 많이 생략되었습니다.

리눅스 프로그래밍 기초 : 시스템 호출, 라이브러리 함수별 실습

ISBN : 978-89-7914-473-4

연습 문제 해답

※ 코드의 양을 줄이기 위해 대부분의 예제 프로그램에서 에러 처리가 많이 생략되었습니다.

1장. 리눅스의 기본 개념과 프로그램 작성

1. 예제 수행

2. 셸 프롬프트 상에서
% man ls
를 실행하여 ls에 대한 매뉴얼을 확인한다.

3. 수정 후 코드

one.c	two.c	printmsg.h

#include <stdio.h>
#include "printmsg.h"
 	
main()
{
printmsg();
}
	#include <stdio.h>
	void printmsg(void)
	{
	printf("hello world!₩n");
	}
		void printmsg(void);

※printmsg.h는 one.c와 동일한 디렉터리에 존재해야 한다.

4. 셸 프롬프트 상에서
% gcc -o one one.c two.c
를 실행한다.

5. 1~10까지 더하는 프로그램의 소스 코드 파일의 이름을 test.c라고 하고 실행 파일의 이름을 test라고 한다.
% gdb test
를 실행하여 디버거를 실행한다.
디버거의 프롬프트가 보이면 아래와 같이 프로그램을 실행한다.
(gdb) start
프로그램이 실행되면 step 명령을 사용하여 라인 단위로 프로그램을 실행을 한다.
(gdb) step
프로그램을 라인별로 실행하다가 특정 변수의 값을 알고자 할 때 print 명령을 사용한다.
(gdb) print sum ← 변수 sum의 값을 출력해본다.
프로그램 실행이 완료되면 quit명령으로 gdb를 종료한다.
(gdb) quit

6. Makefile의 내용

```
one: one.o two.o
        cc -o one one.o two.o
one.o: one.c printmsg.h
        cc -c one.c
two.c: two.c
        cc -c two.c
또는
one: one.c two.c printmsg.h
        cc -o one one.c two.c
```

※ 코드의 양을 줄이기 위해 대부분의 예제 프로그램에서 에러 처리가 많이 생략되었습니다.

2장. 파일 시스템

1. cherry의 절대 경로
/home/cherry
현재 디렉터리가 /root/text일 때의 cherry의 상대 경로
../../home/cherry

2. 먼저 아래처럼 cat을 사용하여 100바이트의 파일을 만든다.
% cat > 100byte.txt ※ 줄을 바꾸기 위해 엔터를 입력하면
123456789 1바이트 크기의 개행 문자가 입력된다.
123456789
123456789
123456789
123456789
123456789
123456789
123456789
123456789
123456789
% cat 100byte.txt 100byte.txt > 200byte.txt
를 실행하여 200바이트의 파일을 생성한다.
% cat 100byte.txt 200byte.txt > 300byte.txt
를 실행하여 300바이트의 파일을 생성한다.

3. test.txt라는 파일에 대해서 권한 설정이 “rw-r----x”와 같아야 하므로
% chmod 641 test.txt
를 실행하면 된다.

4. 로그인한 후에
% ls -l /dev/pts
를 실행하여 열거되는 파일을 확인한다.

※ 코드의 양을 줄이기 위해 대부분의 예제 프로그램에서 에러 처리가 많이 생략되었습니다.

3장. 파일 다루기

1. 교재 본문에 나온 함수 설명을 참고하여 에러 처리 코드를 추가한다.

2. 사용자의 홈 디렉터리가 /home/user 라고 가정한다.

```
#include <stdlib.h>
#include <fcntl.h>

int main()
{
    int filedes;
    filedes = open("~/temp0.txt", O_CREAT | O_RDWR, 0644);
    //filedes = open("/home/user/temp0.txt", O_CREAT | O_RDWR, 0644);

    close(filedes);
}
```

3. 임의의 크기를 가지는 파일을 준비한 상태에서 다음의 코드를 실행한다.

```
% cat test.txt
AAAAA
%
```

```
#include <fcntl.h>
#include <unistd.h>

int main()
{
        int fd;
        char *str = "BBB";

        fd = open("./test.txt", O_RDWR);
        write(fd, str, strlen(str));

        close(fd);
}
```

```
[실행결과]
% cat test.txt
BBBAA
%
```

4. 다음의 실행 코드를 실행해본다. test.txt 파일은 연습문제 3의 것을 사용한다.

```
#include <fcntl.h>
#include <unistd.h>

int main()
{
        int fd, newpos;

        fd = open("./test.txt", O_RDONLY);

        newpos = lseek(fd, (off_t)10, SEEK_END);

        printf("newpos is %d₩n", newpos);

        close(fd);
}
```

```
[실행결과]
% ./04
newpos is 16
%
```

5. read나 write를 호출하고 난 직후에 아래처럼 lseek을 사용해본다.

```
pos = lseek(fd, (off_t)0, SEEK_CUR);
printf("pos is %d₩n", pos);
```

※ 코드의 양을 줄이기 위해 대부분의 예제 프로그램에서 에러 처리가 많이 생략되었습니다.

6. 재귀적인 호출을 사용하여 지정한 디렉터리를 삭제할 수 있다. unlink는 일반 파일을 삭제하는데 사용하고 remove는 비어 있는 디렉터리를 삭제하는데 사용한다.

7.

```
#include <fcntl.h>
#include <unistd.h>

int main(int argc, char *argv[])
{
        int fd;
        char buffer[1024];
        int nread, cnt, numChar = 0;

        if(argc < 2)
                return;

        if((fd = open(argv[1], O_RDONLY)) == -1)
        {
                printf("file open error₩n");
                return;
        }

        while((nread = read(fd, buffer, 1024)) > 0)
        {
                for(cnt = 0; cnt < nread; cnt++)
                {
                        if((buffer[cnt] >= 'a' && buffer[cnt] <= 'z') ||
                           (buffer[cnt] >= 'A' && buffer[cnt] <= 'Z'))
                                numChar++;
                }
        }

        close(fd);

        printf("number of alphabet character is %d₩n", numChar);
}
```

8. read를 호출하여 buffer를 채우고 난 후, 소문자를 대문자로 변경한다. lseek을 사용하여 read로 읽은 바이트만큼 읽기/쓰기 포인터를 앞쪽으로 옮기고 buffer의 내용을 write한다.

```
#include <fcntl.h>
#include <unistd.h>

int main(int argc, char *argv[])
{
        int fd;
        char buffer[1024];
        int nread, cnt, numChar = 0;

        if(argc < 2)
                return;

        if((fd = open(argv[1], O_RDWR)) == -1)
        {
                printf("file open error₩n");
                return;
        }

        while((nread = read(fd, buffer, 1024)) > 0)
        {
                for(cnt = 0; cnt < nread; cnt++)
                {
                        if(buffer[cnt] >= 'a' && buffer[cnt] <= 'z')
                                buffer[cnt] = buffer[cnt] - 'a' + 'A';
                }
                lseek(fd, (off_t)-nread, SEEK_CUR);
                write(fd, buffer, nread);
        }

        close(fd);

        printf("number of alphabet character is %d₩n", numChar);
}
```

※ 코드의 양을 줄이기 위해 대부분의 예제 프로그램에서 에러 처리가 많이 생략되었습니다.

9. read/write를 사용할 때 지정하는 버퍼의 크기가 달라지는 것에 주의하면 된다.

```
#include <fcntl.h>
#include <unistd.h>

int main(int argc, char *argv[])
{
        int fd;
        char buffer;
        int nread, cnt, numChar = 0;

        if(argc < 2)
                return;

        if((fd = open(argv[1], O_RDWR)) == -1)
        {
                printf("file open error₩n");
                return;
        }

        while(read(fd, &buffer, 1) > 0)
        {
                if(buffer >= 'a' && buffer <= 'z')
                        buffer = buffer - 'a' + 'A';
                lseek(fd, (off_t)-1, SEEK_CUR);
                write(fd, &buffer, 1);
        }

        close(fd);
}
```

10. 텍스트 파일을 개방하여 파일의 내용을 변경하는 프로그램을 작성한 후에 백그라운드 모드로 실행한다. 그리고 이때 cat을 사용하여 파일의 내용을 출력해본다.

```
% cat 10.txt
AAA
BBB
% ./10 &
[2] 16917
% cat 10.txt
ZZZ
BBB
% fg
./10
Ctrl+C 입력
%
```

```
#include <fcntl.h>
#include <unistd.h>

int main()
{
        int fd;
        char *buf[64];

        fd = open("./10.txt", O_CREAT | O_RDWR, 0644);

        write(fd, "ZZZ", 3);

        pause();        /* 프로세스를 일시 정지시킨다. */

        close(fd);
}
```

※파일의 내용이 변경되었다.

11. read가 -1을 반환하는 것은 함수의 기본 기능을 수행하는데 있어서 문제가 있었음을 의미한다. 반면에 0을 반환하는 것은 함수의 기본 기능을 제대로 수행하였으나 파일로부터 읽을거리가 없었다..라는 의미를 가진다. 그래서 read가 -1을 반환했다는 것은 읽으려는 시도 자체가 실패했다고 보면 된다. read가 -1을 리턴하는 경우는 % man read를 실행하여 "ERRORS" 부분에 기술되어 있다.

12.

※ 코드의 양을 줄이기 위해 대부분의 예제 프로그램에서 에러 처리가 많이 생략되었습니다.

4장. 파일 관리

1. 교재 본문에 나온 함수 설명을 참고하여 에러 처리 코드를 추가한다.

2. umask를 사용하여 새로운 값을 적용하는 프로그램을 작성한다. 프로그램의 이름이 ex02라고 가정하고 다음과 같은 순서로 실행하여 확인할 수 있다.

```
% umask
0002
% ./ex02
oldmask is 2
newmask is 333
% umask
0002
%
```

```
#include <fcntl.h>
#include <unistd.h>
#include <sys/types.h>

int main()
{
        int fd;
        mode_t oldmask, newmask;

        newmask = 0333;
        oldmask = umask(newmask);

        printf("oldmask is %o₩n", oldmask);
        printf("newmask is %o₩n", newmask);
}
```

3. 일반 유저는 chown을 사용할 수 없으나, 슈퍼 유저는 사용 가능하다.

4. 프로그램의 이름을 ex04라고 가정한다.

```
% ls -l a.txt b.txt
-rw-rw-r--  1 kimyh kimyh 0 Nov 14 21:32 a.txt
lrwxrwxrwx  1 kimyh kimyh 5 Nov 14 21:32 b.txt -> a.txt
% ex04
Ok!!
% rm a.txt
% ex04
Sorry!!
% touch a.txt
% ex04
Ok!!
%
```

```
#include <unistd.h>

int main()
{
        if(!access("./b.txt", F_OK))
                printf("Ok!!₩n");
        else
                printf("Sorry!!₩n");
}
```

5. 소프트 링크는 경로를 사용하여 원본 파일을 가리킨다. 리눅스에서 모든 파일은 파일 시스템이 다르더라도 그 경로가 루트 디렉터리부터 시작하기 때문에 경로를 사용하여 모든 파일을 가리킬 수 있다.
하드 링크는 기존의 파일에 새로운 이름을 하나 추가한다고 생각할 수 있는데 파일의 이름은 하나의 파일 시스템 내에서 유일한 아이노드 블록 번호를 사용하여 아이노드 블록과 연결되어 있다. 파일 이름이 다른 파일 시스템에 있는 아이노드 블록을 가리킬 수 없으므로 하드 링크는 동일한 파일 시스템 내에서만 만들 수 있다.

6. 소프트 링크에 대해 적용하는 일반적인 작업은 소프트 링크가 가리키는 원본 파일에 적용이 되나, readlink는 지정한 파일의 데이터 블록의 내용을 그대로 가져온다.

※ 코드의 양을 줄이기 위해 대부분의 예제 프로그램에서 에러 처리가 많이 생략되었습니다.

```
% ln a.txt hardlink.txt
% ls -l a.txt hardlink.txt
-rw-rw-r--  2 kimyh kimyh 0 Nov 14 21:33 a.txt
-rw-rw-r--  2 kimyh kimyh 0 Nov 14 21:33 hardlink.txt
% ./06
`城?@n [illegible]0?[illegible]x?@煜??  [illegible]??@煜?Q[illegible] ? 릉 4 삯 y?□릉 [illegible] [illegible]>??4?[illegible]Linuxceco
2.6.9-42.EL넒
              ?Dg? ? 카 ??F?  ?A?뤢?튡8[illegible]@    텐휠?4?? 왜4?삯 쿎쬗T?쮪 ?컵 [illegible]
P?텐T?쀅?퓙?웹 ?     ??[illegible]h?영? 닾 ??.?.?[illegible]  ?.?   玉 ?[illegible]???원솟O.적 玉 맞
 ??  [illegible]?[illegible]젳 曺? [illegible]?[illegible]?잴[illegible]?퓙[illegible]T?曺
[illegible]4□[illegible]?[illegible] ?[illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible]?[illegible] ?[illegible]      ?[illegible]
```

```c
#include <unistd.h>

int main()
{
        char buffer[1024];
        int nread;

        nread = readlink("hardlink.txt", buffer, 1024);
        write(1, buffer, nread);
}
```

7. 2개 이상의 파일 시스템(파티션)이 있는 시스템에서 실행해야 한다.

※ 코드의 양을 줄이기 위해 대부분의 예제 프로그램에서 에러 처리가 많이 생략되었습니다.

5장. 디렉터리 다루기

1. opendir 함수나 chdir 함수를 사용하여 간단하게 검사할 수 있다. 지정한 경로의 디렉터리를 개방하거나 지정한 경로로 변경할 때 에러가 발생하면 해당 경로의 디렉터리가 없다고 생각할 수 있다.

```
#include <unistd.h>
#include <dirent.h>
#include <sys/types.h>

int isDir(char *);

int main(int argc, char *argv[])
{
        if(argc != 2)
                return 0;

        if(isDir(argv[1]))
                printf("%s not exist₩n", argv[1]);
        else
                printf("%s exist₩n", argv[1]);
}

int isDir(char *path)
{
        DIR *dirp;
        dirp = opendir(path);
        closedir(dirp);
        return dirp == NULL ? 1 : 0;
}
```

2. opendir로 디렉터리를 개방하고 readdir로 디렉터리 엔트리를 하나씩 읽어온다. 디렉터리 엔트리가 디렉터리인지 일반 파일인지를 1번 문제의 함수를 사용하여 검사한다.

```
#include <unistd.h>
#include <dirent.h>
#include <sys/types.h>

int isDir(char *);

int main(int argc, char *argv[])
{
        DIR *dirp;
        struct dirent *dentry;
        int num_dir = 0, num_fil = 0;

        if(argc != 2)
                return 0;

        if((dirp = opendir(argv[1])) == NULL)
                return 0;

        while(dentry = readdir(dirp))
        {
                if(dentry->d_ino != 0)
                {
                        if(isDir(dentry->d_name) == 0)
                                num_dir++;
                        else
                                num_fil++;
                }
        }

        printf("%d dirs, %d files₩n", num_dir, num_fil);

        closedir(dirp);
}

int isDir(char *name)
{
        DIR *dirp;
        dirp = opendir(name);
        closedir(dirp);
        return dirp == NULL ? 1 : 0;
}
```

4. readdir 함수로 디렉터리 엔트리를 읽어오고, stat 함수를 사용하여 디렉터리 엔트리가 가리키는 파일의 블록

※ 코드의 양을 줄이기 위해 대부분의 예제 프로그램에서 에러 처리가 많이 생략되었습니다.

수를 알아온다.

```
struct entry {
        char ent_name[512];                     // readdir 함수로 알 수 있다.
        blkcnt_t st_blocks;             // stat 함수로 알 수 있다.
};
```

```
struct entry list[1024];                        // 이와 같은 배열을 만들어 저장한 후 정렬한다.
```

5. 4장의 readlink 함수를 사용한다. 지정한 파일에 readlink 함수를 적용했을 때 반환되는 값이 -1이 아니면 소프트 링크이다.

```
#include <unistd.h>

int main(int argc, char *argv[])
{
        char name[1024];
        int nread;

        if(argc != 2)
                exit(1);

        nread = readlink(argv[1], name, 1024);  // 이 부분을 함수로 구현한다.

        printf("%s is %s a soft-link₩n", argv[1], nread == -1 ? "not" : "");
}
```

6. 하드링크는 원본 파일에 새로운 이름을 하나 더 추가하는 것이다. stat이나 fstat 함수로 파일의 정보를 가져왔을 때 하드링크 카운트의 값이 2 이상이면 하드링크이다.

```
struct stat 형식의 구조체에서 nlink_t 형식의 st_nlink의 값이 2이상인지 검사한다.
```

※ 코드의 양을 줄이기 위해 대부분의 예제 프로그램에서 에러 처리가 많이 생략되었습니다.

6장. 프로세스 관리

1. 유닉스/리눅스의 wc 명령어는 표준 입력되는 내용의 라인 수, 단어 수, 글자 수를 출력한다. 메시지를 출력하는 명령어의 이름이 "hello" 라면...

```
$ hello | wc -c
```

wc의 -c 옵션은 글자(character) 수를 출력한다.

2. 루트 디렉터리를 제외한 모든 파일과 디렉터리는 부모 디렉터리가 존재한다. 마찬가지로 init 프로세스를 제외한 모든 프로세스도 부모 프로세스가 존재한다. 하나의 프로세스는 부모 프로세스를 오직 하나만 가지며 부모 프로세스는 여러 개의 자식 프로세스를 가질 수 있다. 다음과 같이 ps 명령어를 -ef 옵션과 함께 사용하면 시스템의 모든 프로세스를 상세한 정보와 함께 나열해 볼 수 있다.

```
$ ps -ef
```

자신의 소유로 된 프로세스만을 보려면 grep 명령어를 자신의 사용자명(id)을 인자로 사용하여 걸러주면 된다.

```
$ ps -ef | grep 'kimyh'
root     26659 26650  0 14:07 ?        00:00:00 login -- kimyh
kimyh    26664 26659  0 14:07 pts/58   00:00:00 -bash
kimyh    28637 26664  0 14:46 pts/58   00:00:00 ps -ef
kimyh    28638 26664  0 14:46 pts/58   00:00:00 grep kimyh
```

위의 예에서 'kimyh'란 사용자와 관련된 프로세스는 첫 번째 줄을 제외한 나머지 3개의 줄이다. 각각의 프로세스 번호(PID)는 26664, 28637, 28638이고 각각의 부모 프로세스 번호(PPID)는 26659, 26664, 26664 이다. 자신의 프로세스의 부모 프로세스를 찾고, 다시 부모 프로세스의 부모 프로세스를 찾는다. 이렇게 거슬러서 부모의 부모를 찾아가면 결국은 init 프로세스를 만나게 된다.

3. '$ cat > test.txt' 와 같이 실행된 후에 블록킹 되는 명령어를 후면 처리로 여러 개 실행한 후에 2번 문제와 같이 자신이 소유하고 있는 프로세스를 확인한다. 확인된 프로세스 번호를 사용하여 kill 명령어를 수행한다.

※ 코드의 양을 줄이기 위해 대부분의 예제 프로그램에서 에러 처리가 많이 생략되었습니다.

7장. 프로세스 다루기 (1) : 생성과 종료

1. 사용자로부터 숫자를 하나 입력 받은 후에 fork를 사용하여 자식 프로세스를 생성하고, 각자 맡은 일을 수행한다.

```
#include <unistd.h>
#include <sys/types.h>

int main()
{
        pid_t pid;
        int num, i, sum, mul;

        printf("input a number : ");
        scanf("%d", &num);

        pid = fork();

        if(pid > 0)       // parent
        {
                for(i = 1, sum = 0; i <= num; i++)
                        sum += i;
                printf("sum is %d₩n", sum);
        }
        else              // child
        {
                for(i = 1, mul = 1; i <= num; i++)
                        mul *= i;
                printf("mul is %d₩n", mul);
        }
}
```

2. 부모 프로세스가 두 개의 자식 프로세스를 생성한다.
부모 프로세스는
    pid1 > 0 && pid2 > 0
    1초를 쉰 뒤에 (sleep 후에) num++;을 수행하여 1을 출력하고,
두 개의 자식 프로세스는
    pid1 == 0 || pid2 == 0
    num = num + 2;를 수행하여 2를 출력한다.
자식 프로세스를 만드는데 많은 시간이 들지 않는다면 2 2 1이 순서대로 출력될 것이다.

3. head는 지정한 파일들의 내용 중 앞의 일부만을 보여주는 명령어이다. 먼저 a.txt, b.txt, c.txt 파일을 준비한 상태에서 $ head a.txt b.txt c.txt 를 실행하면 세 파일의 앞 부분이 출력되는 것을 볼 수 있을 것이다. exec 계열의 함수를 사용하여 head를 실행한다. 이때 인자로 a.txt b.txt c.txt를 넣으면 된다.

```
        char *arg[] = {"head", "a.txt", "b.txt", "c.txt", (char *)0};
        …
        execl("/usr/bin/head", "head", "a.txt", "b.txt", "c.txt", (char *)0);
        …
        execlp("head", "head", "a.txt", "b.txt", "c.txt", (char*)0);
        …
        execv("/usr/bin/head", arg);
        …
        execvp("head", arg);
```

4. 부모 프로세스는 인자의 개수만큼 (파일의 수만큼) fork를 실행하여 자식 프로세스를 수행한다.

※ 코드의 양을 줄이기 위해 대부분의 예제 프로그램에서 에러 처리가 많이 생략되었습니다.

```
#include <unistd.h>
#include <sys/types.h>

int main(int argc, char *argv[])
{
        pid_t pid;
        char filename[64];
        int n;

        for(n = 1; n < argc; n++)
        {
                strcpy(filename, argv[n]);
                printf("run ₩"wc %s₩"₩n", filename);
                pid = fork();
                if(pid == 0)     // child
                {
                        execlp("wc", "wc", filename, (char *)0);
                        //printf("wc %s₩n", filename);
                        exit(1);          // if fail to run execlp
                }
        }
}
```

5.

```
#include <unistd.h>
#include <fcntl.h>

int filedes;
void final(void);

int main()
{
        ssize_t nread;
        char buffer[1024];

        atexit(final);

        filedes = open("a.txt", O_RDONLY);
        while((nread = read(filedes, buffer, 1024)) > 0)
        {
                printf("%s", buffer);
        }

        //exit(1);
}


void final(void)
{
        printf("close filedes₩n");
        close(filedes);
}
```

※ 코드의 양을 줄이기 위해 대부분의 예제 프로그램에서 에러 처리가 많이 생략되었습니다.

8장. 프로세스 다루기 (2) : 동기화, 속성, 환경 변수

1.

```
#include <unistd.h>
#include <sys/types.h>

int main()
{
        int x, y, i;
        pid_t pid;
        int c_result = 0, p_result = 1;

        printf("input two numbers (1~10) : ");
        scanf("%d %d", &x, &y);

        pid = fork();

        if(pid > 0)
        {
                for(i = 0; i < y; i++)
                        p_result *= x;
                printf("parent : %d₩n", p_result);
                wait(&c_result);
                c_result = c_result >> 8;
                printf("result is %d₩n", p_result + c_result);
        }
        else if(pid == 0)
        {
                for(i = x; i <= y; i++)
                        c_result += i;
                printf("child : %d₩n", c_result);
                exit(c_result);
        }
}
```

2.

```
#include <unistd.h>
#include <sys/types.h>

int main()
{
        pid_t pid;

        pid = fork();

        if(pid > 0)
        {
                printf("parent process₩n");
                printf("pid:%d, ppid:%d₩n", getpid(), getppid());
        }
        else if(pid == 0)
        {
                printf("child process₩n");
                printf("pid:%d, ppid:%d₩n", getpid(), getppid());
        }
}
```

3.

※ 코드의 양을 줄이기 위해 대부분의 예제 프로그램에서 에러 처리가 많이 생략되었습니다.

```
#include <unistd.h>
#include <sys/types.h>

#define NUM      5

int main()
{
        pid_t pid[NUM];
        int status;

        int n;

        for(n = 0; n < NUM; n++)
        {
                pid[n] = fork();
                if(pid[n] == 0)        // 갓 생성된 자식 프로세스라면..
                        break;
        }

        if(n == NUM)     // 부모 프로세스라면...
        {
                for(n = 0; n < NUM; n++)
                {
                        wait(pid[n], &status, 0);
                        printf("child %d - exit(%d)₩n", pid[n], status);
                }
        }
        else if(n < NUM)          // 자식 프로세스 중 하나라면..
        {
                sleep(n);
                exit(n);
        }
}
```

4. 자식 프로세스는 생성되자마자 setsid()를 사용하여 새로운 세션을 만들어 리더가 된다. 그 후에 10초마다 현재 시각을 출력하면 된다.

```
#include <unistd.h>
#include <time.h>
#include <sys/types.h>
#include <fcntl.h>

int main()
{
        time_t result;
        pid_t pid;

        int filedes;
        char buffer[32];

        pid = fork();

        if(pid > 0)
        {
                sleep(1);
                exit(1);
        }
        else if(pid == 0)
        {
                setsid();
                filedes = open("./curtime.txt", O_RDWR | O_CREAT, 0644);
                for(;;)
                {
                        result = time(NULL);
                        strcpy(buffer, asctime(localtime(&result)));
                        printf("%s", buffer);
                        write(filedes, buffer, strlen(buffer));
                        sleep(10);
                }
        }

}
```

5.

※ 코드의 양을 줄이기 위해 대부분의 예제 프로그램에서 에러 처리가 많이 생략되었습니다.

[05.c]

```
#include <unistd.h>

int main()
{
        printf("[[[begin of 05.c]]]₩n");
        putenv("MYDATA=5");
        execl("./05-1", "05-1", (char *)0);
        printf("[[[end if 05.c]]]₩n");
}
```

[05-1.c]

```
#include <unistd.h>

extern char **environ;

int main()
{
        printf("[[[begin of 05-1.c]]]₩n");
        printf("MYDATA=%s₩n₩n", getenv("MYDATA"));

        while(*environ)
                printf("%s₩n", *environ++);

        printf("[[[end of 05-1.c]]]₩n");
}
```

※ 코드의 양을 줄이기 위해 대부분의 예제 프로그램에서 에러 처리가 많이 생략되었습니다.

9장. 프로세스 간 통신

1. 다음이 시그널은 키보드를 통해서 입력될 수 있는 것들로, 키보드의 키를 눌러 발생할 수 있는 것과 그렇지 않은 것이 있다.

키보드를 통해 발생되는 시그널 : SIGHUP, SIGINT, SIGQUIT
입출력 터미널과 관련된 시그널 : SIGSTOP, SIGTTIN, SIGTTOU, SIGTSTP

2. 네임드 파이프는 두 개의 프로세스 사이에서 데이터를 전달해주는 파이프이다. 임시로 생성되었다가 삭제되는 익명 파이프(anonymous pipe)와는 달리 'rm' 명령어로 삭제하지 않는 한 항상 파일로서 존재한다.

①'ls -al > np' 명령어 라인은 np로 자신의 표준 출력을 넣어주는 프로세스가 있지만 표준 입력을 np로부터 가져오는 프로세스가 없다. 그래서 np로부터 표준 입력을 취하는 프로세스가 생길 때까지 'ls -al' 은 블록킹된다.

②위의 ①에서 실행한 'ls -al'의 실행 결과가 'cat'에 의해서 표준 출력된다.

③위에서 설명함

※ 코드의 양을 줄이기 위해 대부분의 예제 프로그램에서 에러 처리가 많이 생략되었습니다.

10장. 시그널과 시그널 처리

1.

```
#include <unistd.h>
#include <signal.h>
#include <fcntl.h>

int flag = 1;

main()
{
        static struct sigaction act;

        void int_handler(int);

        act.sa_handler = int_handler;
        sigfillset(&(act.sa_mask));
        sigaction(SIGINT, &act, NULL);

        while(flag)
        {
                printf("in loop...₩n");
                sleep(1);
        }
}

void int_handler(int signum)
{
        int fd;
        char *msg = "interrupted by SIGINT...₩n";

        fd = open("01_out.txt", O_CREAT | O_RDWR, 0644);
        write(fd, msg, strlen(msg));
        close(fd);

        flag = 0;
}
```

2. signal 함수

```
#include <signal.h>

typedef void (*sighandler_t)(int);

sighandler_t signal(int signum, sighandler_t handler);
```

signum으로 지정한 시그널에 대해서 handler로 지정한 방식으로 대처한다. handler로 적용할 수 있는 값은 sigaction과 마찬가지로 SIG_IGN, SIG_DFL, 함수에 대한 포인터가 될 수 있으며, 각각의 의미는 sigaction에서 쓰는 것과 같이 무시, 기본 동작 수행, 지정한 함수 수행이 된다. signal과 sigaction의 차이점은 signal로 설정한 “특정 시그널에 대한 대응 방법”이 1회성이이지만, sigaction은 한번 설정하면 그 효과가 계속 적용된다.

※ 코드의 양을 줄이기 위해 대부분의 예제 프로그램에서 에러 처리가 많이 생략되었습니다.

3.

```
/* ex10-5.c */
#include <unistd.h>
#include <signal.h>

main()
{
        sigset_t set;
        int count = 0;

        sigfillset(&set);
        //sigaddset(&set, SIGINT);

        sigprocmask(SIG_BLOCK, &set, NULL);

        while(count < 10)
        {
                if(count < 5)
                        printf("don't distrub me (%d)₩n", count);
                else
                {
                        if(count == 5)
                        {
                                sigemptyset(&set);
                                sigaddset(&set, SIGINT);
                                sigprocmask(SIG_UNBLOCK, &set, NULL);
                        }
                        printf("disturb me ..₩n");
                }
                sleep(1);
                count++;
        }

}
```

4. 부모 프로세스는 pause()를 사용하여 자식 프로세스가 보내는 종료 신호를 캐치한다.

```
/* ex10-06.c */
#include <signal.h>
#include <sys/types.h>

main()
{
        pid_t pid;
        int count = 5;

        if((pid = fork()) > 0)
        {
                printf("[parent] hello₩n");
                pause();
                printf("[parent] bye!₩n");
        }
        else if(pid == 0)
        {
                printf("[child] hello₩n");
                sleep(1);
                kill(getppid(), SIGINT);
                printf("[child] bye₩n");
        }
        else
                printf("fail to fork₩n");
}
```

5. alarm 함수는 지정한 시간이 지난 후에 자기 자신에게 SIGALRM 시그널을 전달한다. sleep 함수는 alarm 함수를 사용하여 구현됐는데, alarm 함수가 반복적으로 사용된다고 볼 수 있다. 예를 들어 sleep(1)이라고 사용하면 1초 뒤에 SIGALRM 함수가 발생하고 다시 alarm(1)을 실행한다고 보면 된다. sleep 함수는 지정한 시간을 주기로 alarm 함수를 반복해서 호출한다고 보면 된다.

※ 코드의 양을 줄이기 위해 대부분의 예제 프로그램에서 에러 처리가 많이 생략되었습니다.

11장. 파이프를 이용한 통신

1. 부모 프로세스는 자식 프로세스를 생성하기 전에 두 개의 파이프를 생성한다. 자식 프로세스를 생성하면 각 프로세스는 사용하지 않는 파이프 관련 기술자를 닫아주고(close), 파이프를 통해 메시지를 주고 받으면 된다.

```
#include <unistd.h>
#include <fcntl.h>
#include <sys/types.h>

main()
{
        pid_t pid[2];
        int n;

        int p1[2], p2[2];

        char msg[64];

        if(pipe(p1) == -1 || pipe(p2) == -1)
        {
                printf("fail to call pipe()₩n");
                exit(1);
        }

        for(n = 0; n < 2; n++)
        {
                pid[n] = fork();
                if(pid[n] == 0)
                        break;
        }

        if(n == 2)      // parent process
        {

                close(p1[0]);
                close(p2[0]);

                strcpy(msg, "hello, child1₩n");
                write(p1[1], msg, strlen(msg)+1);

                strcpy(msg, "hello, child2₩n");
                write(p2[1], msg, strlen(msg)+1);
        }
        else if(n == 0) // 1st child
        {
                close(p1[1]);
                close(p2[0]);
                close(p2[1]);

                read(p1[0], msg, 64);
                printf("%s", msg);
        }
        else if(n == 1) //2nd child
        {
                close(p1[0]);
                close(p1[1]);
                close(p2[1]);

                read(p2[0], msg, 64);
                printf("%s", msg);
        }
}
```

※ 코드의 양을 줄이기 위해 대부분의 예제 프로그램에서 에러 처리가 많이 생략되었습니다.

2. 하나의 프로세스(부모 프로세스)가 다 수의 프로세스(3개의 자식 프로세스)로부터 파이프를 통해 메시지를 수신하게 된다. 자식 프로세스들이 보낸 메시지들의 도착 순서를 알 수 없으므로 부모 프로세스는 select()를 사용하여 자식 프로세스와 연결된 파이프를 감시하고, 읽을거리가 생기면 해당 파이프 기술자를 통해서 메시지를 읽는다.

```
#include <unistd.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/time.h>
#include <sys/wait.h>

#define MSGSIZE 16
#define NC      3

int onerror(char *msg)
{
        printf("%s", msg);
        exit(1);
}

main()
{
        pid_t pid[NC];
        int n, r, c;

        int filedes[NC][2];
        char msg[MSGSIZE];

        fd_set initset, newset;

        for(n = 0; n < NC; n++)
                if(pipe(filedes[n]) == -1)
                        onerror("fail to call pipe()₩n");

        for(n = 0; n < NC; n++)
        {
                pid[n] = fork();
                if(pid[n] == 0)
                        break;
        }

        if(n == NC)      // parent process
        {
                printf("parent: %d₩n", getpid());

                for(r = 0; r < NC; r++)
                        close(filedes[r][1]);

                FD_ZERO(&initset);
                FD_SET(filedes[0][0], &initset);
                FD_SET(filedes[1][0], &initset);
                FD_SET(filedes[2][0], &initset);

                newset = initset;

                while(select(filedes[2][0] + 1, &newset, NULL, NULL, NULL) > 0)
                {
                        if(FD_ISSET(filedes[0][0], &newset))
                                if(read(filedes[0][0], msg, MSGSIZE) > 0)
                                        printf("[parent] %s from child1₩n", msg)
;
                        if(FD_ISSET(filedes[1][0], &newset))
                                if(read(filedes[1][0], msg, MSGSIZE) > 0)
                                        printf("[parent] %s from child2₩n", msg)
;
                        if(FD_ISSET(filedes[2][0], &newset))
                                if(read(filedes[2][0], msg, MSGSIZE) > 0)
                                        printf("[parent] %s from child3₩n", msg)
;
                        newset = initset;
                }

        }

        // 아래에 계속...
```

※ 코드의 양을 줄이기 위해 대부분의 예제 프로그램에서 에러 처리가 많이 생략되었습니다.

```
        else if(n == 0) // 1st child
        {
                printf("child1 : %d₩n", getpid());

                for(r = 0; r < NC; r++)
                        close(filedes[r][0]);
                close(filedes[1][1]);
                close(filedes[2][1]);

                for(r = 0; r < 3; r++)
                {
                        sleep((r + 1) % 4);
                        printf("child1 : send message %d₩n", r);
                        write(filedes[0][1], "i'm child1", MSGSIZE);
                }

                printf("child1 : bye!₩n");
                exit(0);
        }
        else if(n == 1) //2nd child
        {
                printf("child2 : %d₩n", getpid());

                for(r = 0; r < NC; r++)
                        close(filedes[r][0]);
                close(filedes[0][1]);
                close(filedes[2][1]);

                for(r = 0; r < 3; r++)
                {
                        sleep((r + 3) % 4);
                        printf("chlid2 : send message %d₩n", r);
                        write(filedes[1][1], "i'm child2", MSGSIZE);
                }

                printf("child2 : bye!₩n");
                exit(0);
        }
        else if(n == 2)
        {
                printf("child3 : %d₩n", getpid());

                for(r = 0; r < NC; r++)
                        close(filedes[r][0]);
                close(filedes[0][1]);
                close(filedes[1][1]);

                for(r = 0; r < 3; r++)
                {
                        sleep((r + 5) % 4);
                        printf("child3 : send message %d₩n", r);
                        write(filedes[2][1], "i'm child3", MSGSIZE);
                }

                printf("child3 : bye!₩n");
                exit(0);
        }
}
```

3. '$ man popen'을 실행하여 매뉴얼을 참고한다.

4. 파이프의 원자화 동작

원자화 동작이란 어떤 동작(operation)이 어떤 이유에서건 방해 받지 않고 처리 되는 것을 의미한다. 파이프를 통해서 프로세스가 메시지를 주고받을 때 한 번에 보내거나 또는 한 번에 받는 메시지의 크기가 파이프의 크기를 벗어나지 않으면 원자화 동작으로 처리가 되나, 메시지가 파이프의 크기를 넘게 되면 파이프의 크기만큼 잘려져서 전송된다. 이것은 원자화 동작이 아니기 때문에 잘려진 메시지 중 다음 것을 파이프로 쓰거나 읽을 때 방해를 받을 수 있다. 방해를 받으면 잘려진 메시지의 뒷부분은 전달되지 않을 수 있다. 확실한 동작을 보장하려면 파이프를 통해 전송하려는 메시지의 크기는 항상 파이프의 크기보다 같거나 작아야 한다. 파이프의 크기는 /usr/include/linux/limits.h에서 확인하거나 pathconf 함수를 사용하여 확인할 수 있다.

```
#define PIPE_BUF        4096    /* # bytes in atomic write to a pipe */
```

※ 코드의 양을 줄이기 위해 대부분의 예제 프로그램에서 에러 처리가 많이 생략되었습니다.

5. pathconf 함수를 사용하여 시스템에서 설정된 파이프의 최대 크기를 확인한 후에 이 크기를 파이프로 메시지를 쓰거나 읽을 때 사용하는 버퍼의 크기로 사용한다.

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>

main()
{
        int fd_read, fd_write, p[2];
        pid_t pid;

        char *buffer;
        char *file1 = "a.txt";
        char *file2 = "b.txt";

        int pipe_size, nread;

        pipe_size = pathconf(file1, _PC_PIPE_BUF);
        printf("pipe size is %d₩n", pipe_size);

        pipe(p);
        pid = fork();

        if(pid > 0)     // parent
        {
                printf("hello! (%d)₩n", getpid());
                fd_read = open(file1, O_RDONLY);
                buffer = (char *)(malloc(sizeof(char) * pipe_size));
                while((nread = read(fd_read, buffer, pipe_size)) > 0)
                        write(p[1], buffer, nread);
                close(fd_read);
                free(buffer);
        }
        else if(pid == 0)       // child
        {
                printf("hello! (%d)₩n", getpid());
                fd_write = open(file2, O_RDWR | O_CREAT, 0644);
                buffer = (char *)(malloc(sizeof(char) * pipe_size));
                while((nread = read(p[0], buffer, pipe_size)) > 0)
                        write(fd_write, buffer, nread);
                close(fd_write);
                free(buffer);
        }
        else
                ;
}
```

※ 코드의 양을 줄이기 위해 대부분의 예제 프로그램에서 에러 처리가 많이 생략되었습니다.

12장. 소켓을 이용한 통신 (1) : 연결 지향형 모델

1. 바이너리 형태의 인터넷 주소가 저장되어 있는 in_addr_t형의 값은 4바이트의 크기를 가진다. 이를 하위 바이트부터 차례대로 찍어본다.

```
#include <arpa/inet.h>
#include <unistd.h>

main()
{
        char *address = "197.1.2.3";

        in_addr_t ipaddr;
        int m, n, mask = 255;

        printf("size of in_addr_t is %d₩n", sizeof(in_addr_t));

        ipaddr = inet_addr(address);

        printf("sizeof int is %d₩n", sizeof(int));

        for(n = 0; n < sizeof(int); n++)
        {
                //printf("%d₩n", ipaddr);
                m = (ipaddr >> (n * 8)) & mask;
                printf("%d(%x) ", m, m);
        }
        printf("₩n");
}
```

2. 서버 프로그램을 실행하면 listen을 실행하고 난 후에 10초를 대기하다가 accept를 호출하여 클라이언트의 연결을 수락한다. accept를 호출하기 전 10초 동안에 backlog가 허용한 수 만큼의 클라이언트는 connect를 호출하고 난 후에 반환 값을 바로 받게 되나 나머지 클라이언트는 반환 값을 바로 받지 않는다.

```
% ./server 10 &   //backlog의 값을 10으로 하여 서버 프로그램을 백그라운드로 실행한다.
% ./client 15 &   //서버에 연결을 시도하는 프로세스15개를 동시에 생성한다.
```

15개의 클라이언트 프로그램 중에 backlog가 허용하는 수 만큼의 클라이언트는 connect를 호출하고 반환값을 바로 받지만 나머지 클라이언트는 바로 받지 못한다. connect 호출의 반환값을 바로 받은 클라이언트 중 일부가 종료해서 backlog에 여유가 생길 때 반환값을 받게 된다.

※실제로 테스트를 해보면.. 실제 적용되는 backlog의 값은 listen함수를 실행하면서 넣은 인자 값보다 평균 3정도 더 크다. 가령 backlog를 10으로 적용하여 서버를 실행한 상태에서 20개의 클라이언트를 생성해보면 13개의 클라이언트가 connect의 반환 값을 바로 받고 나머지 7개는 13개의 클라이언트 중 일부가 종료해야 반환 값을 받게 된다.

※ 코드의 양을 줄이기 위해 대부분의 예제 프로그램에서 에러 처리가 많이 생략되었습니다.

① 서버 프로그램

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>

#define SIZE     sizeof(struct sockaddr_in)

main(int argc, char *argv[])
{
        int sockfd_listen, sockfd_connect;
        int ret, backlog;

        struct sockaddr_in server = {AF_INET, 5000, INADDR_ANY};

        if(argc != 2)
        {
                printf("usage error₩n");
                exit(1);
        }

        backlog = atoi(argv[1]);

        sockfd_listen = socket(AF_INET, SOCK_STREAM, 0);
        ret = bind(sockfd_listen, (struct sockaddr *)&server, SIZE);
        if(ret == -1)
                printf("[server] error on bind₩n");

        printf("[server] listen₩n");
        listen(sockfd_listen, backlog);

        sleep(10);

        while(1)
        {
                printf("[server] accept!₩n");
                sockfd_connect = accept(sockfd_listen, NULL, NULL);
                printf("[server] accepted (%d)₩n", sockfd_connect);
                //sleep(1);
                close(sockfd_connect);
                printf("[server] connection lost₩n");
        }

}
```

※ 코드의 양을 줄이기 위해 대부분의 예제 프로그램에서 에러 처리가 많이 생략되었습니다.

② 클라이언트 프로그램

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>

#define SIZE     sizeof(struct sockaddr_in)
#define NUM      64

main(int argc, char *argv[])
{
        int sockfd, ret_val;
        struct sockaddr_in server = {AF_INET, 5000};
        pid_t pid[NUM];
        int np, limit;

        if(argc != 2)
        {
                printf("usage error₩n");
                exit(1);
        }
        limit = atoi(argv[1]);

        for(np = 0; np < limit; np++)
        {
                pid[np] = fork();
                if(pid[np] == 0)
                        break;
        }

        server.sin_addr.s_addr = inet_addr("127.0.0.1");

        printf("[client(%d,%d)] socket₩n", np, getpid());
        sockfd = socket(AF_INET, SOCK_STREAM, 0);
        if(sockfd == -1)
                printf("[client(%d,%d)] fail to call socket()₩n", np, getpid());

        printf("[client(%d,%d)] connect₩n", np, getpid());
        ret_val = connect(sockfd, (struct sockaddr *)&server, SIZE);

        printf("[client(%d,%d)] ret_val is %d₩n", np, getpid(), ret_val);
        if(ret_val == 0)
                close(sockfd);
}
```

※ 코드의 양을 줄이기 위해 대부분의 예제 프로그램에서 에러 처리가 많이 생략되었습니다.

3.

① server 프로그램

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <signal.h>

#define SIZE    sizeof(struct sockaddr_in)
#define MSGLEN  1024

void closesock(int sig);

int sockfd_connect;

main()
{
        int sockfd_listen, nread;
        char msg[MSGLEN];
        struct sockaddr_in server = {AF_INET, 5000, INADDR_ANY};
        struct sigaction act;

        act.sa_handler = closesock;
        sigfillset(&(act.sa_mask));
        sigaction(SIGPIPE, &act, NULL);

        if((sockfd_listen = socket(AF_INET, SOCK_STREAM, 0)) == -1)
        {
                printf("fail to call socket()₩n");
                exit(1);
        }

        if(bind(sockfd_listen, (struct sockaddr *)&server, SIZE) == -1)
        {
                printf("fail to call bind()₩n");
                exit(1);
        }

        if(listen(sockfd_listen, 5) == -1)
        {
                printf("fail to call listen()₩n");
                exit(1);
        }

        while(1)
        {
                if((sockfd_connect = accept(sockfd_listen, NULL, NULL)) == -1)
                {
                        printf("fail to call accept()₩n");
                        continue;
                }
                printf("accepted₩n");

                while((nread = recv(sockfd_connect, msg, MSGLEN, 0)) > 0)
                        send(sockfd_connect, msg, nread, 0);

                printf("close(sockfd_connect)₩n");
                close(sockfd_connect);
        }
}

void closesock(int sig)
{
        close(sockfd_connect);
        printf("connection is lost₩n");
        exit(0);
}
```

※ 코드의 양을 줄이기 위해 대부분의 예제 프로그램에서 에러 처리가 많이 생략되었습니다.

② client 프로그램

```
#include <ctype.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <stdio.h>

#define SIZE     sizeof(struct sockaddr_in)
#define MSGLEN   1024
main()
{
        int sockfd, nread;
        char send_c[1024], recv_c[1024];
        struct sockaddr_in server = {AF_INET, 5000};

        server.sin_addr.s_addr = inet_addr("127.0.0.1");

        if((sockfd = socket(AF_INET, SOCK_STREAM, 0)) == -1)
        {
                printf("fail to call socket()₩n");
                exit(1);
        }

        if(connect(sockfd, (struct sockaddr *)&server, SIZE) == -1)
        {
                printf("fail to call connect()₩n");
                exit(1);
        }

        recv_c[0] = '₩0';
        while(1)
        {
                if(strlen(recv_c) == 0)
                        printf("Input a message₩n");
                fgets(send_c, MSGLEN, stdin);

                send(sockfd, send_c, strlen(send_c), 0);

                if((nread = recv(sockfd, recv_c, MSGLEN, 0)) > 0)
                {
                        printf("%s₩n", recv_c);
                }
                else
                {
                        printf("server has no reply₩n");
                        close(sockfd);
                        exit(1);
                }
        }
}
```

4. 클라이언트는 자신이 보낼 메시지를 보관하는 버퍼와 서버가 되돌려준 메시지를 보관하는 버퍼를 따로 선언하여 사용하므로, 서버가 되돌려준 메시지를 받자마자 이 둘의 크기를 비교하면 된다.

```
                if((nread = recv(sockfd, recv_c, MSGLEN, 0)) > 0)
                {
                        recv_c[nread] = '₩0';
                        if(strlen(recv_c) == strlen(send_c))
                                printf("[O] ");
                        else
                                printf("[X] ");
                        printf("%s₩n", recv_c);
                }
                else
                {
                        ...
                }
```

5. 예제 프로그램 ex12-10을 사용하여 실험해보면, 첫 번째 클라이언트가 연결된 상태에서 두 번째 클라이언트의 연결 요청이 수락되지 않는 것을 확인할 수 있다. 서버는 클라이언트의 연결 요청을 수락하기 위해서 accept 함수를 호출해야 하는데 위의 경우에서 서버는 첫 번째 클라이언트와 recv, send 함수를 사용하여 데이터를 주고 받는 상태, 즉 accept를 호출한 상태가 아니기 때문에 두 번째 클라이언트의 connect를 사용한 연결 요청은 블록킹 되어 있게 된다. 첫 번째 클라이언트가 연결을 해제하게 되면 서버는 다시 accept를 호출하게 되고 이때 두 번째

※ 코드의 양을 줄이기 위해 대부분의 예제 프로그램에서 에러 처리가 많이 생략되었습니다.

클라이언트의 연결 요청이 수락된다.

6.

① 서버 프로그램

```
#include <sys/types.h>
#include <sys/socket.h>
#include <fcntl.h>
#include <netinet/in.h>
#include <signal.h>
#include <string.h>

#define SIZE     sizeof(struct sockaddr_in)
#define MSGLEN   1024

void closesock(int sig);

int sockfd_connect;

main()
{
        int sockfd_listen, nread, filelength, fd, recv_size;
        char msg[MSGLEN], filename[256];
        struct sockaddr_in server = {AF_INET, 5000, INADDR_ANY};
        struct sigaction act;

        act.sa_handler = closesock;
        sigfillset(&(act.sa_mask));
        sigaction(SIGPIPE, &act, NULL);

        if((sockfd_listen = socket(AF_INET, SOCK_STREAM, 0)) == -1)
        {
                printf("fail to call socket()₩n");
                exit(1);
        }

        if(bind(sockfd_listen, (struct sockaddr *)&server, SIZE) == -1)
        {
                printf("fail to call bind()₩n");
                exit(1);
        }

        if(listen(sockfd_listen, 5) == -1)
        {
                printf("fail to call listen()₩n");
                exit(1);
        }

        if((sockfd_connect = accept(sockfd_listen, NULL, NULL)) == -1)
        {
                printf("fail to call accept()₩n");
                exit(1);
        }
        printf("accepted₩n");

        recv(sockfd_connect, filename, 256, 0);
        recv(sockfd_connect, &filelength, sizeof(filelength), 0);

        strcat(filename, ".backup");
        printf("%s, %d₩n", filename, filelength);

        recv_size = 0;
        fd = open(filename, O_RDWR | O_CREAT, 0644);

        while((nread = recv(sockfd_connect, msg, MSGLEN, 0)) > 0 &&
                recv_size <= filelength)
        {
                recv_size += nread;
                write(fd, msg, nread);
        }

        printf("close(sockfd_connect)₩n");
        close(sockfd_connect);
}

void closesock(int sig)
{
        close(sockfd_connect);
        printf("connection is lost₩n");
        exit(0);
}
```

※ 코드의 양을 줄이기 위해 대부분의 예제 프로그램에서 에러 처리가 많이 생략되었습니다.

② 클라이언트 프로그램

```
#include <ctype.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/stat.h>
#include <netinet/in.h>
#include <stdio.h>

#define SIZE     sizeof(struct sockaddr_in)
#define MSGLEN   1024
main(int argc, char *argv[])
{
        int sockfd, nread, fd;
        struct stat finfo;
        char buffer[MSGLEN];

        char send_c[1024], recv_c[1024];
        struct sockaddr_in server = {AF_INET, 5000};

        char filename[256];
        int filelength;

        if(argc != 2)
        {
                printf("usage error₩n");
                exit(1);
        }

        strcpy(filename, argv[1]);
        fd = open(filename, O_RDONLY);
        fstat(fd, &finfo);
        filelength = finfo.st_size;

        server.sin_addr.s_addr = inet_addr("127.0.0.1");

        if((sockfd = socket(AF_INET, SOCK_STREAM, 0)) == -1)
        {
                printf("fail to call socket()₩n");
                exit(1);
        }

        if(connect(sockfd, (struct sockaddr *)&server, SIZE) == -1)
        {
                printf("fail to call connect()₩n");
                exit(1);
        }

        send(sockfd, filename, 256, 0);
        send(sockfd, &filelength, sizeof(filelength), 0);

        while((nread = read(fd, buffer, MSGLEN)) > 0)
        {
                send(sockfd, buffer, nread, 0);
        }

        close(sockfd);
        close(fd);
}
```

7. 바이트 순서 (byte order)

2바이트 이상의 크기를 가지는 데이터를 메모리에 저장하는 방식은 CPU의 종류에 따라 다르며 Big-Endian 방식과 Little-Endian 방식으로 나뉜다. Big-Endian 방식은 데이터의 상위 바이트가 메모리에 먼저 저장되고(메모리 주소상 작은 주소), Little-Endian 방식은 데이터의 하위 바이트가 메모리에 먼저 저장된다. 0x1234라는 값을 Big-Endian 방식으로 저장하면 상위 바이트가 먼저 메모리에 저장되므로 0x12, 0x34의 순서로 바이트 값이 저장되나 Little-Endian 방식은 0x34, 0x12의 순서로 바이트 값이 저장된다. 만약에 Big-Endian 방식을 사용하는 시스템이 전송한 0x1234란 데이터를 Little-Endian이 수신하게 되면 0x3412란 값이 될 것이다.

CPU에 따른 메모리 상에서의 저장 방식을 호스트 바이트 순서 (host byte order)라고 부르며 호스트 바이트 순서가 CPU에 따라 서로 다르더라도 네트워크를 통한 데이터 전송에서는 미리 정해진 표준 방식을 따르는데 이를 네트워크 바이트 순서 (network byte order)라고 부르며 Big-Endian 방식을 따른다. 그래서 호스트 바이트 순서가 Little-Endian인 시스템은 데이터를 전송할 때 Little-Endian으로 저장된 값을 Big-Endian으로 변환한여 전송해야 한다.

관련 함수 : htonl, htons, ntohl, ntohs

※ 코드의 양을 줄이기 위해 대부분의 예제 프로그램에서 에러 처리가 많이 생략되었습니다.

13장. 소켓을 이용한 통신 (2) : 비연결 지향형 모델과 관련 함수

1. /usr/include/netinet/in.h 파일 안에 struct in_addr 구조체형이 선언되어 있다.

```
/* Internet address.  */
typedef uint32_t in_addr_t;
struct in_addr
  {
    in_addr_t s_addr;
  };
```

```
#include <unistd.h>
#include <netdb.h>

in_addr_t gethostaddr(char *hostname);

main(int argc, char *argv[])
{
        in_addr_t addr;

        if(argc != 2)
                exit(1);

        addr = gethostaddr(argv[1]);
        printf("%s₩n", inet_ntoa(addr));

}

in_addr_t gethostaddr(char *hostname)
{
        struct hostent *hent;
        struct in_addr *addr_n;
        char **ptr;

        if((hent = gethostbyname(hostname)) == NULL)
                return -1;

        if(hent->h_addrtype == AF_INET)
        {
                ptr = hent->h_addr_list;
                addr_n = (struct in_addr *)(*ptr);
                return (in_addr_t)(addr_n->s_addr);
        }

        return 0;
}
```

2. 예제 프로그램 ex13-04s.c와 ex13-04c.c를 사용하여 하나의 서버를 실행한 상태에서 두 개 이상의 클라이언트를 실행한 후에 메시지를 입력하면, 각각의 클라이언트가 송신한 메시지를 서버가 정상적으로 수신하는 것을 확인할 수 있다. TCP를 이용한 방법은 클라이언트가 connect를 실행하여 서버쪽에 연결을 요청하면 서버는 accept를 호출하여 연결 요청을 수락하게 되고 이때부터 서로 메시지를 주고 받을 수 있게 된다. 12장의 예제 프로그램은 이미 하나의 클라이언트와 연결되어 메시지를 주고 받는 서버는 또다른 클라이언트의 연결 요청을 받을 수 없는 상태이기 때문에 서버는 한번에 하나의 클라이언트와 메시지를 주고받을 수 있다. 반면에 UDP를 사용한 13장의 예제는 서버와 클라이언트가 연결 작업 없이 바로 메시지를 주고 받을 수 있다.

※ 코드의 양을 줄이기 위해 대부분의 예제 프로그램에서 에러 처리가 많이 생략되었습니다.

3.

① 서버 프로그램

```
/* Server */
#include <time.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>

#define SIZE     sizeof(struct sockaddr_in)
#define MSGSIZE 128

main()
{
        int sockfd;
        char msg, msg_time[MSGSIZE];
        time_t curtime;

        struct sockaddr_in server = {AF_INET, 2007, INADDR_ANY};

        struct sockaddr_in client;
        int client_len = SIZE;

        if((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) == -1)
        {
                printf("fail to call socket()₩n");
                exit(1);
        }

        if(bind(sockfd, (struct sockaddr *)&server, SIZE) == -1)
        {
                printf("fail to call bind()₩n");
                exit(1);
        }

        while(1)
        {
                if(recvfrom(sockfd, &msg, 1, 0, (struct sockaddr *)&client, &client_len) == -1)
                {
                        printf("fail to receive message₩n");
                        continue;
                }

                curtime = time(NULL);
                strcpy(msg_time, asctime(localtime(&curtime)));
                if(sendto(sockfd, msg_time, strlen(msg_time), 0, (struct sockaddr *)&client, client_len)
== -1)
                {
                        printf("fail to send message₩n");
                        continue;
                }
        }
}
```

※ 코드의 양을 줄이기 위해 대부분의 예제 프로그램에서 에러 처리가 많이 생략되었습니다.

② 클라이언트 프로그램

```
/* Client */
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>

#define SIZE     sizeof(struct sockaddr_in)
#define MSGSIZE 128

main()
{
        int sockfd;
        char msg = '₩n', msg_time[MSGSIZE];

        struct sockaddr_in client = {AF_INET, INADDR_ANY, INADDR_ANY};

        int server_len = SIZE;
        struct sockaddr_in server = {AF_INET, 2007};
        server.sin_addr.s_addr = inet_addr("202.31.200.87");

        if((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) == -1)
        {
                printf("fail to call socket()₩n");
                exit(1);
        }

        if(sendto(sockfd, &msg, 1, 0, (struct sockaddr *)&server, server_len) == -1)
        {
                printf("fail to send message₩n");
                exit(1);
        }

        if(recvfrom(sockfd, msg_time, MSGSIZE, 0, (struct sockaddr *)&server, &server_len) == -1)
        {
                printf("fail to receive message₩n");
                exit(1);
        }

        printf("%s", msg_time);
}
```

※ 코드의 양을 줄이기 위해 대부분의 예제 프로그램에서 에러 처리가 많이 생략되었습니다.

4. 기존의 방식과 비교하여 서버 프로그램은 수정할 부분이 없다. 클라이언트 쪽에서 connect()를 호출하는 부분이 추가하면 된다.

① 서버 프로그램

```
/* Server */

#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>

#define SIZE     sizeof(struct sockaddr_in)

main()
{
        int sockfd;
        char msg, prev;

        struct sockaddr_in server = {AF_INET, 2007, INADDR_ANY};

        struct sockaddr_in client;
        int client_len = SIZE;

        if((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) == -1)
        {
                printf("fail to call socket()₩n");
                exit(1);
        }

        if(bind(sockfd, (struct sockaddr *)&server, SIZE) == -1)
        {
                printf("fail to call bind()₩n");
                exit(1);
        }

        prev = '₩n';
        while(1)
        {
                if(recvfrom(sockfd, &msg, 1, 0, (struct sockaddr *)&client, &client_len) == -1)
                {
                        printf("fail to receive message₩n");
                        continue;
                }


                printf("%s%c", (prev == '₩n') ? "[recv] " : "", msg);
                prev =  msg;

                if(sendto(sockfd, &msg, 1, 0, (struct sockaddr *)&client, client_len) == -1)
                {
                        printf("fail to receive message₩n");
                        continue;
                }
        }
}
```

※ 코드의 양을 줄이기 위해 대부분의 예제 프로그램에서 에러 처리가 많이 생략되었습니다.

② 클라이언트 프로그램

```
/* Client */

#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>

#define SIZE     sizeof(struct sockaddr_in)

main()
{
        int sockfd;
        char msg, prev;

        struct sockaddr_in client = {AF_INET, INADDR_ANY, INADDR_ANY};

        int server_len = SIZE;
        struct sockaddr_in server = {AF_INET, 2007};
        server.sin_addr.s_addr = inet_addr("202.31.200.87");

        if((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) == -1)
        {
                printf("fail to call socket()₩n");
                exit(1);
        }

        prev = '₩n';

        if(connect(sockfd, (struct sockaddr *)&server, server_len) == -1)
        {
                printf("fail to call connect()₩n");
                exit(1);
        }

        while(read(0, &msg, 1) != 0)
        {
                //if(sendto(sockfd, &msg, 1, 0, (struct sockaddr *)&server, server_len) == -1)
                if(send(sockfd, &msg, 1, 0) == -1)
                {
                        printf("fail to send message₩n");
                        continue;
                }

                //if(recvfrom(sockfd, &msg, 1, 0, (struct sockaddr *)&server, &server_len) == -1)
                if(recv(sockfd, &msg, 1, 0) == -1)
                {
                        printf("fail to receive message₩n");
                        continue;
                }

                printf("%s%c", (prev == '₩n') ? "[recv] " : "", msg);
                prev = msg;
        }
}
```

※ 코드의 양을 줄이기 위해 대부분의 예제 프로그램에서 에러 처리가 많이 생략되었습니다.

5.

① 서버 프로그램

```
/* Server */
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <fcntl.h>

#define SIZE     sizeof(struct sockaddr_in)
#define MSGLEN   1024

main()
{
        int sockfd, fd, nread, filelen;
        char buffer[MSGLEN], filename[256];

        struct sockaddr_in server = {AF_INET, 2007, INADDR_ANY};

        struct sockaddr_in client;
        int client_len = SIZE;

        if((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) == -1)
        {
                printf("fail to call socket()₩n");
                exit(1);
        }

        if(bind(sockfd, (struct sockaddr *)&server, SIZE) == -1)
        {
                printf("fail to call bind()₩n");
                exit(1);
        }

        recv(sockfd, filename, 256, 0);
        recv(sockfd, &filelen, sizeof(filelen), 0);

        strcat(filename, ".backup");
        printf("[server] %s, %d₩n", filename, filelen);

        fd = open(filename, O_RDWR | O_CREAT, 0644);

        while((nread = recv(sockfd, buffer, MSGLEN, 0)) == MSGLEN)
        {
                write(fd, buffer, nread);
        }
        write(fd, buffer, nread);

        printf("close(sockfd)₩n");
        close(sockfd);

        close(fd);
}
```

※ 코드의 양을 줄이기 위해 대부분의 예제 프로그램에서 에러 처리가 많이 생략되었습니다.

② 클라이언트 프로그램

```
/* Client */
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <stdio.h>
#include <fcntl.h>
#include <sys/stat.h>

#define SIZE     sizeof(struct sockaddr_in)
#define MSGLEN   1024

main(int argc, char *argv[])
{
        int sockfd, fd, nread;
        struct stat finfo;
        char buffer[MSGLEN];

        struct sockaddr_in client = {AF_INET, INADDR_ANY, INADDR_ANY};
        int server_len = SIZE;
        struct sockaddr_in server = {AF_INET, 2007};

        char filename[256];
        int filelen;

        if(argc != 2)
        {
                printf("usage error₩n");
                exit(1);
        }

        strcpy(filename, argv[1]);
        fd = open(filename, O_RDONLY);
        fstat(fd, &finfo);
        filelen = finfo.st_size;

        server.sin_addr.s_addr = inet_addr("202.31.200.87");

        // 에러가 발생하지 않는다고 가정해서, 에러처리하지 않음
        sockfd = socket(AF_INET, SOCK_DGRAM, 0);
        connect(sockfd, (struct sockaddr *)&server, server_len);

        send(sockfd, filename, 256, 0);
        send(sockfd, &filelen, sizeof(filelen), 0);

        while((nread = read(fd, buffer, MSGLEN)) > 0)
        {
                if(send(sockfd, buffer, nread, 0) == -1)
                        printf("fail to send message₩n");
        }

        close(sockfd);
        close(fd);
}
```